Pledge on honor

1. Carefully read the text below

2. Once you have understood this text, and agreed with it, recopy it in the text field below.

3. Once you have recopied this text, you can proceed to the end of the quizz and to the final exam

Pledge on honor

CPU kernel mode vs. administrator user

Select all correct claims below. These claims concern the CPU kernel / supervisor mode and the administrator user (root, admin) of an Operating System.

- An admin / root user can execute privileged assembly instructions without using system calls
- The applications of an admin / root are executed in kernel mode only
- An admin / root can install new O.S. features (e.g. drivers, system libraries) in an Operating System
- An admin / root can upgrade an Operating System

Deadlock

Select the correct following claims. These claims concern the following pseudo code, assuming P1 and P2 are started at the same time, with a=0.

Process P1:

mutex_lock(m1)

mutex_lock(m2)

a++

mutex_unlock(m2)

mutex_unlock(m1)

Process P2:

mutex_lock(m2)

mutex_lock(m1)

a--

mutex_unlock(m1)

mutex_unlock(m2)

- "a" can have a negative value
- A deadlock can occur
- P1 and P2 can modify a at the same time
- m1 and m2 can never be locked together
- Mutual exclusion on "a" is not enforced by this program

DMA - Direct Memory Access

Select all correct claims below.

- DMA controllers can access the main memory of a computer system

Using DMA is efficient for large data transfer

DMA is efficient for small data transfer

A user-level application can select whether its data should be transferred by DMA or not

DMA-enabled devices can use DMA to transfer their data to main memory

Inter Process Communication with pipes

Select all the correct assertions about the following bash command executed, for instance, in Solaris (just like during labs):

$ vmstat −s | grep fork

This command creates two processes

This command creates one process

A unidirectional pipe is created by this command from "vmstat" to "grep"

"vmstat" must complete before "grep" starts executing

The output of "vmstat" is sent to "grep" only once all the output of vmstat has been produced

Protection mechanisms

What are the minimum hardware mechanisms on which an Operating System must rely to ensure protection between user processes?

Privileged instructions of CPU kernel mode

Memory Management Unit

Direct Memory Access Controller

Hardware Timer

RTOS - Interrupt Service Routines

The following questions are related to the Interrupt Service Routines (ISR) of Real-Time Operating Systems. Select all correct claims.

Real-Time tasks have a higher priority than ISR

ISR are expected to be short to allow urgent tasks to execute as soon as possible

ISR can be interrupted by urgent tasks

ISR can be split into two parts to allow urgent task to be executed sooner

ISR can be enabled / disabled by user-level code

Software signals

Check all the correct claims which concern the code below.

The following code makes it possible to exchange signals between a sender and a receiver. We assume that the receiver is started a few seconds before the sender. Also, the command line to start the sender provides the process id of receiver. Last but not least, we assume that all works as expected (no process is killed during execution, etc.)

Receiver code:

```
void getSignal(int signo) {
```

```
  if (signo == SIGUSR1) {
    printf("Received SIGUSR1\n");
  } else {
    printf("Received%d\n", signo);
  }
}

int main(void) {
  printf("Registering SIGUSR1 signal / #SIGUSR1=%d\n", SIGUSR1);
  signal(SIGUSR1, getSignal);
  sleep(30);
}
```

Sender code:

```
int main(int argc, char**argv) {

  int pid;
  if (argc <2) {
    printf("Usage: sender <destination process pid>\n");
    exit(-1);
  }
  pid = atoi(argv[1]);
  printf("Sending SIGURG to %d\n", pid);
  kill(pid, SIGURG);
  printf("Sending SIGUSR1 to %d\n", pid);
  kill(pid, SIGUSR1);
  printf("Sending SIGUSR1 to %d\n", pid);
  kill(pid, SIGUSR1);
 }
```

- getSignal() is called three times
- The return of all system calls are checked for errors
- getSignal() is called two times
- getSignal() is called one time
- If SIGKILL were to be sent at first by sender instead of SIGURG, the behavior of receiver would be the same.

Swapping

Select all the true following claims about process swapping from main memory to disk, and vice-versa.

- Swapping exchanges processes from one CPU core to another one
- Swapping is more likely to be used when the main memory is close to be totally allocated
- Swapping can be performed by the Memory Management Unit
- Processes cannot execute when they have been swapped out
- Swapping improves execution time of processes

System calls vs. functions of libraries

Which following claims are correct? Thee claims are related to the differences between system calls and functions of libraries.

- Performing a call to a functions of a library is faster than performing a call to a system call
- System calls can execute privileged assembly instructions but functions of libraries cannot
- The manual pages of system calls are listed in a different section than the ones of library functions
- Function of libraries cannot call memory allocations routines while system calls can

There are more system calls than functions of libraries

User-level vs. Kernel-level threads

Select all the following correct claims.

Kernel threads are scheduled by the OS kernel

User-level threads are scheduled by the OS kernel

User-level threads can be scheduled with a scheduling policy which is different from the one of the OS

Java threads are always handled by the OS and not by the JVM

Average Waiting Time - FCFS

Compute the Average Waiting Time of the following set of tasks scheduled with the First-Come-First-Served policy.

Set of tasks

Task	Arrival time	Computation time
T1	5	3
T2	3	2
T3	6	4
T4	2	2

Answer

Average Waiting Time - SJF

Compute the Average Waiting Time of the following set of tasks scheduled with the Shortest Job First policy.

Set of tasks

Task	Arrival time	Computation time
T1	5	3
T2	3	5
T3	6	4
T4	2	3

Answer

Interactive systems

In interactive systems, processes using all their quantum of time are penalized by being granted a low priority.

- True
- False

Page Fault

Can this code provoke a page fault?

```
int cpt;

int main() {

  int a[1];

  while (1) {

    a[cpt --] = 1;

  }

}
```

- True
- False

Envoyer